CHRONOWSKI Amaury
JOBERT–ROLLIN Gabin

# Rapport du projet de TSI

Année universitaire 2021-2022

# Introduction

Dans le cadre du module de TSI, nous avons réalisé un jeu de tir à la première personne (FPS). On se déplace dans un labyrinthe défini au préalable où des cibles sont disposées. Notre travail était de s'appuyer sur la base d'un programme qui disposait déjà de fonctions permettant de gérer et d'afficher des objets pour faire notre jeu. Nous avons fait en sorte de créer le labyrinthe, gérer les collisions entre les éléments, modéliser le titre du joueur par un rayon et faire un affichage basique.

# Déroulement d'une partie

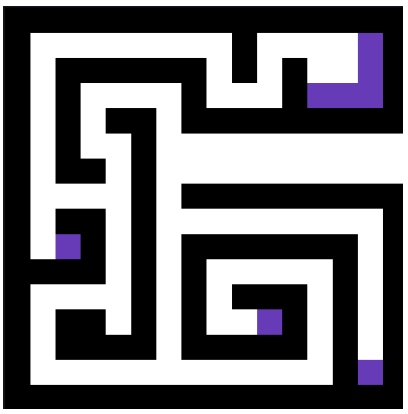
Huit stégosaures sont cachés dans le labyrinthe, le but est de les trouver et de leurs tirer dessus pour les faire disparaître.
Une fois le programme lancé, nous nous trouvons à l'écran d'accueil, cliquez n'importe où sur l'écran pour commencer le jeu. Voici les contrôles :

| Action | Touche |
|---|---|
| Avancer | W |
| Reculer | S |
| Gauche | A |
| Droite | D |
| Tirer | Clic Gauche |
| Contrôle caméra | Bougez la sourie |

Une fois les huit stégosaures touchés, le compteur s'arrête et le joueur ne peut plus bouger, on pourrait facilement installer une fonction qui remettrait la partie à zéro.

# Création du labyrinthe

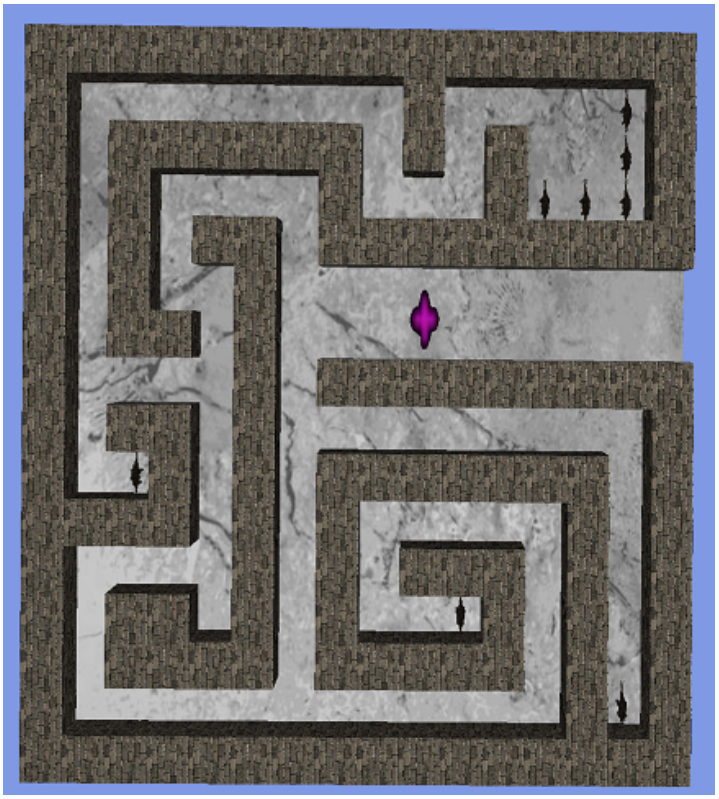
Tout d'abord nous nous somme mis d'accord sur la forme du labyrinthe que l'on peut voir ci-dessous (le noir represente les murs, le violet les cibles et le blanc le vide:



Ce labyrinthe n’est qu’un exemple, notre programme peut en réalité construire n’importe quel labyrinthe où l’on segmente chaque élément sur une grille de 16x16. On renseigne la disposition des éléments sur un fichier .txt, les 0 représentent le vide,les 1 représentent les murs et le 2 représentent les cibles, ci-dessous la forme du fichier utilisé pour faire le labyrinthe (vous pouvez tester en modifiant le fichier stage.txt) :

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 1 0 0 0 0 2 1
1 0 1 1 1 1 1 1 0 1 0 1 0 0 2 1
1 0 1 0 0 0 0 1 0 0 0 1 2 2 2 1
1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1
1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1
1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1
1 0 2 1 0 1 0 1 1 1 1 1 1 1 0 1
1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 1
1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1
1 0 1 1 0 1 0 1 0 0 2 1 0 1 0 1
1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

On utilise le même élément plusieurs fois, via une boucle for qui parcourt une liste de liste puis chaque liste contient une ligne d'élément. Après, on place le sol qui fait la même taille que le labyrinthe.De plus, on place la caméra sur un bord vide du labyrinthe. (La croix au mileux est le curseur du joueur)

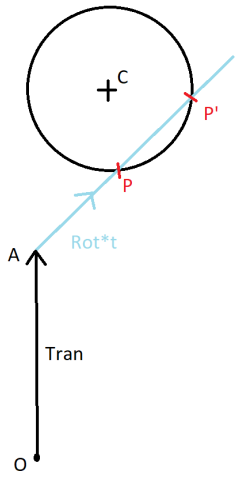


# Collisions

Notre système de collisions se base sur les coordonnées des objets. On récupère les futures coordonnées du joueur et on regarde s'il se trouve à l'intérieur d'un mur ou d'une cible et empêche le déplacement.

# Système de tir (Ray Casting)

Le système de tir a été géré par Ray casting, en prenant comme hitbox une sphère. Comme le montre le schéma ci-dessous, le centre de la sphère est noté C, le joueur A, l'origine O. Les matrices contenant les transformations du joueur sont appelées "Trans" (Contenant la translation), et Rot contenant le vecteur directionnel correspondant aux angles d'euler.




Le rayon touche seulement si il existe au moins un point P qui est à la fois sur la trajectoire du rayon, et qui appartient au cercle. En mettant cette condition en équation et en la réarrangeant on arrive à une équation du second degré.

$$\begin{aligned} \|\overrightarrow{PC}\| &= R^2 \\ \overrightarrow{PC}.\overrightarrow{PC} &= R^2 \\ (\overrightarrow{PA}+\overrightarrow{AC})(\overrightarrow{PA}+\overrightarrow{AC}) &= R^2 \\ \|\overrightarrow{PA}\|^2 + 2\overrightarrow{AC}.\overrightarrow{PA} + \|\overrightarrow{AC}\|^2 &= R^2 \\ t^2 + 2\frac{\overrightarrow{AC}.\overrightarrow{rot}}{\|\overrightarrow{rot}\|^2}t + \frac{\overrightarrow{AC}.\overrightarrow{AC}-R^2}{\|\overrightarrow{rot}\|^2} &= 0 \end{aligned}$$

le rayon touche que si il existe au moins une solution à cette équation, et donc si : $b^2 - 4c \geq 0$ Il suffit donc de déterminer pour chaque stégosaure si cette condition est réalisée.

Mais pour faire ces calculs nous avons eu besoin du vecteur directionnel correspondant aux angles d'Euler de la caméra.

Pour cela nous avons utilisé les équations ci-dessous, résultant de la multiplication, des matrices de rotation pour les 3 angles, yaw, pitch et roll.

$$\begin{aligned} x &= -\sin(yaw) \\ y &= -\sin(pitch)\cos(yaw) \\ z &= -\cos(pitch)\cos(yaw) \end{aligned}$$

# Affichage première personne

A chaque itération de la boucle du jeu, on regarde le mouvement du curseur par rapport à la dernière itération, puis on change le vecteur contenant angles d'euler en fonction. Si le curseur se trouve aux extrémités droite ou gauche de la

fenêtre, celui-ci est directement téléporté de l'autre côté de la fenêtre. Le curseur n'a pas été caché pour montrer ce fonctionnement.

# Améliorations

Cacher le curseur avec :
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);

Installer un écran titre et la possibilitée de recommencer :
Dans viewerGL.py insérer à la fin de "Mouse_Button_callback" à l'emplacement marqué par un commentaire un appel à une fonction qui :
Remettrait le joueur à la position de départ :
self.cam.transformation.translation= [ Coordonnées du Spawn]
remettait à True la variable "visible" des stégosaures

Installer des boutons sur l'écran titre en changeant la fonction "updateMouseCursor" pour